Delay-Differential Equations with Constant Lags

Article · January 2012

DOI: 10.5642/codee.201209.01.10

2 authors:

Lawrence Shampine
Southern Methodist University
252 PUBLICATIONS **12,713** CITATIONS

SEE PROFILE

Skip Thompson
Radford University
43 PUBLICATIONS **1,328** CITATIONS

SEE PROFILE

CODEE Journal

Volume 9 Article 10

5-14-2012

Delay-Differential Equations with Constant Lags

Lawrence Shampine

Skip Thompson

Delay–Differential Equations with Constant Lags

Lawrence F. Shampine
Southern Methodist University

Skip Thompson
Radford University

Keywords: delay differential equations, constant lag
Manuscript received on March 24, 2009; published on May 14, 2012.

Abstract: This article concerns delay–differential equations (DDEs) with constant lags. DDEs increasingly are being used to model various phenomena in mathematics and the physical sciences. For such equations the value of the derivative at any time depends on the solution at a previous "lagged" time. Although solving DDEs is similar in some respects to solving ordinary differential equations (ODEs), it differs in some rather significant ways. These differences are discussed briefly. The effect the differences can have on systems of ODEs and DDEs is illustrated. Popular approaches used in the development of numerical methods for solving DDEs are described. Available MATLAB DDE solvers and a Fortran 90 solver based on these approaches are mentioned. Finally, some pointers to further resources available to interested readers are given.

1 Delayed Effects

Ordinary differential equations (ODEs) are widely used in modeling, but it is becoming more common that models take into account effects that have a delayed action. The populations $y_1(t)$ of prey and $y_2(t)$ of predator are often modeled by a system of first-order ODEs

$$\begin{aligned} y_1'(t) &= a\, y_1(t) + b\, y_1(t)\, y_2(t) \\ y_2'(t) &= c\, y_1(t) + d\, y_1(t)\, y_2(t) \end{aligned}$$

with constants a, b, c, d. Hale [1] considers variations of this model that have a resource limitation on the prey and a birth rate of predators that responds to changes in the populations only after a constant time lag. These models have the form

$$\begin{aligned} y_1'(t) &= a\, y_1(t) \left(1 - \frac{y_1(t)}{m}\right) + b\, y_1(t)\, y_2(t) \\ y_2'(t) &= c\, y_1(t) + d\, y_1(t-\tau)\, y_2(t-\tau) \end{aligned}$$

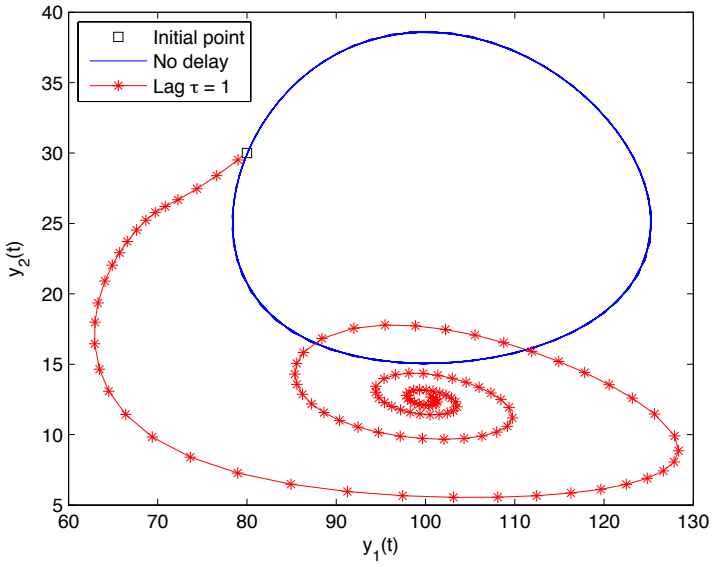

Figure 1: Effect of Delay on Predator-Prey Model.

This is a first-order system of *delay–differential equations* (DDEs). Here $\tau > 0$ is a lag in the effect of population changes and m is another parameter. Only one lag appears in this system, but in general there might be lags $\tau_1, \ldots, \tau_k$. In some applications a lag $\tau_j(t, y)$ might depend on the time t and/or state vector y. New phenomena are possible with these more general lags, so for the sake of simplicity, we consider only constant lags in this article.

2 Effects of Delays

There are important differences between ODEs and DDEs that are easily seen. To be concrete, we display some results for the two kinds of predator-prey models in Figure 1. In this computation the physical parameters were $a = 0.25, b = -0.01, c = -1.00, d = 0.01, m = 200$ and the lag $\tau = 1$. The ODEs were solved on the interval $[0, 100]$ with initial values $y_1(0) = 80, y_2(0) = 30$.

A first important difference between ODEs and DDEs is that *for DDEs we must specify values of the solution not just at the initial point $t = 0$ but also at earlier times.* Specifically, if $0 < t < \tau$, the argument of the terms $y_1(t - \tau)$ and $y_2(t - \tau)$ in the differential equation is less than 0. Values of the solution components for $t \leq 0$ are called the *history* of the solution. The MATLAB [2] ODE solver `ode23` requires users to specify a vector of initial values. Now we understand why the corresponding DDE solver `dde23` must ask for more, namely a history function $h(t)$ that defines the vector $y(t) = h(t)$ for $t \leq 0$. It is common that the history is a constant vector, so as a convenience, the DDE solvers of MATLAB allow users to provide this vector instead of a function. This was done when computing Figure 1 because the DDEs were solved with a constant history equal to the vector of initial values for the ODEs.

The existence and uniqueness of a solution of a system of DDEs,

$$y'(t) = f(t, y(t), y(t-\tau)) \tag{2.1}$$

with history $y(t) = h(t)$ for $t \le 0$ can be deduced from corresponding results for ODEs using the *method of steps*: On the interval $[0, \tau)$, the differential equations (2.1) are a system of ODEs because the term $y(t-\tau)$ is a known function, namely $h(t-\tau)$. Having established the existence of a unique solution on this interval, the same argument applies to the interval $[\tau, 2\,\tau)$, and so forth.

This all sounds fine, but lurking in the details is another important difference between ODEs and DDES: *A solution normally has a jump discontinuity in the first derivative at the initial point.* That is because the derivative from the left is determined by the history function, $y'(0-) = h'(0)$, but the derivative from the right is defined by the differential equation,

$$y'(0+) = f(0, y(0), y(0-\tau)) = f(0, h(0), h(0-\tau))$$

and generally these two derivatives are different. In the case of the predator-prey equations with the parameters and constant history of Figure 1, $y_2'(0-) = 0$ and $y_2'(0+) = (-1)(80) + (0.01)(80)(30) = -56$. Unfortunately, the same argument shows that a lag of τ causes a discontinuity at 0 to propagate to $\tau, 2\tau, 3\tau, \ldots$.

It gets worse if there is more than one lag. A second lag τ_2 implies discontinuities at $\tau_2, 2\tau_2, 3\tau_2, \ldots$, but in addition, it implies that the discontinuity at τ propagates to $\tau+\tau_2, \tau+2\tau_2, \ldots$. Every discontinuity is propagated by both lags, so *the initial discontinuity spawns a whole tree of discontinuities.* As might be expected, discontinuities cause trouble for numerical methods, especially if they are very close to one another. A simple example shows why this is surprisingly common: Suppose there are lags 1 and 1/3. An initial discontinuity is propagated from $t = 0$ to $t = 1$ by the first lag. The second lag successively propagates the discontinuity to 1/3, 2/3, 3/3. The snag is that the lag 1/3 is not represented exactly in a computer, so 1/3 + 1/3 + 1/3 is not exactly equal to 1—it is *extremely* close and therein lies the problem.

For the kinds of DDEs we are talking about, *retarded DDEs*, it is easy to deduce from the differential equations that the order of a discontinuity increases by one each time it propagates. Numerical methods "see" discontinuities only up to a certain order, so in solving retarded DDEs numerically we can ignore all high order discontinuities and in effect, get control of this kudzu vine [3] of discontinuities by hacking off branches close to the stem.

3 Numerical Methods

With the method of steps in mind, it is natural to use popular methods for ODEs like explicit Runge–Kutta (RK) formulas as the basis for programs to solve DDEs. That is what we did with the two solvers of MATLAB and a Fortran 90 program, DDE_SOLVER [4]. However, the approach requires two important extensions of the usual formulas.

3.1 Looking Back: RK+

At first thought it seems straightforward to apply an explicit RK formula to the ODEs of the method of steps, but on second thought, where do the delayed terms in the differential equations come from? An RK method integrates (2.1) by a sequence of steps from an approximation $y_n \approx y(t_n)$ to $y_{n+1} \approx y(t_{n+1})$ at $t_{n+1} = t_n + h$. The method produces an approximate solution *only* at mesh points t_m and generally a delayed argument will fall between mesh points, so how do we approximate the solution there? It is natural to interpolate previously computed approximations to $y(t_m)$, and perhaps the slopes $y'(t_m)$, to approximate $y(t - \tau)$ for an argument $t_j \le t - \tau \le t_{j+1}$. For sufficiently low order formulas like those of `dde23`, cubic Hermite interpolation to value and slope at t_j and t_{j+1} provides approximations throughout $[t_j, t_{j+1}]$ that have the same order of accuracy as the values at the end points. It is harder to do this with formulas of higher order as in DDE_SOLVER. Still, there are formulas for which linear combinations of values of f formed in the course of taking a step, plus perhaps a few more, provide accurate solutions throughout the span of the step. These so-called **continuous extensions** of RK formulas are important when solving ODEs, but they are essential when solving DDEs. An important practical matter is that a DDE solver must save all the information needed to approximate delayed terms. This can amount to a lot of data.

3.2 Breaking the τ Barrier: RK++

When the step size h is bigger than the smallest lag τ, the formula needs an approximate solution at a delayed argument $t - \tau > t_n$. This is a Catch-22 [5]: We need an approximate solution in the span of the current step $[t_n, t_{n+1}]$ before the step is actually taken! To avoid this difficulty, early solvers simply did not use step sizes $h > \tau$, but this means that problems with "small" τ are expensive, perhaps prohibitively so. For example, if we were to solve a DDE with a smallest delay of 0.1 over an interval $[0, 1000]$, we would have to take (at least) 10,000 steps, no matter how smooth the solution is. Using an $h > \tau$ amounts to evaluating an *implicit* RK formula, this despite the basic formula being explicit. This can be done effectively with a scheme rather like the Picard iteration used to prove the existence of solutions of ODEs. The continuous extension on the previous step is used to predict a solution throughout $[t_n, t_{n+1}]$. With this, the explicit RK formula can be evaluated to get a solution at t_{n+1} and a continuous extension formed for the current step. The process is repeated with the continuous extension for the current step until the values at t_{n+1} converge. Typical codes allow only a few iterations before reducing h to speed up the process. It is always possible to get convergence, for if h is reduced to a value smaller than τ, the formula is explicit. Iteration is expensive compared to an explicit step, but it is done only when the behavior of the solution permits a step size big enough to make it worthwhile.

4 Where Can I Learn More?

The Scholarpedia article [6] is a short introduction to DDEs that is a good sequel to this article. A more complete, but still introductory, treatment is found in the text [7],

especially as regards solving DDEs in MATLAB. After that text was published, the `ddesd` program was added to MATLAB to solve DDEs with time- and state-dependent delays. The MATLAB problem–solving environment documents its solvers and their use, but in addition, there is a tutorial with many examples that show how advanced capabilities of the software can be to solve a wide range of problems for DDEs with constant lags. This tutorial and its example programs are available online at [8] along with the Fortran 90 DDE_SOLVER and preprints of several relevant papers. Also available are drivers `ezdde23` and `ezddesd` for the MATLAB solvers with a syntax that some find simpler than `dde23` and `ddesd`, respectively. Links to other publicly available DDE solvers in several languages and computing environments are found at http://twr.cs.kuleuven.be/research/software/delay/software.shtml.

It is nearly as easy to solve DDEs as ODEs in MATLAB, and not much more difficult in Fortran 90 using DDE_SOLVER. The obvious difference is that you have to provide a history and define the lags. And, of course, the differential equations are a little more complicated because of the delayed terms.

References

[1] J. Hale, *Functional Differential Equations.* Springer-Verlag, Berlin, 1978.

[2] MATLAB 7, The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA, 2006.

[3] M. Shores, *The Amazing Story of Kudzu: Love it, or Hate it... It Grows on You!*, http://www.maxshores.com/kudzu/.

[4] S. Thompson and L.F. Shampine, A friendly Fortran DDE solver. *Appl. Numer. Math.* 56:503–516 (2006). See also http://www.radford.edu/~thompson/ffddes/index.html.

[5] J. Heller, *Catch-22.* Classic Ed., Simon & Schuster, New York, 1999.

[6] S. Thompson, Delay-differential equations. Scholarpedia, http://www.scholarpedia.org/article/Delay-differential_equations.

[7] L.F. Shampine, I. Gladwell, and S. Thompson, *Solving ODEs with MATLAB.* Cambridge Univ. Press, New York, 2003.

[8] L.F. Shampine and S. Thompson, Web support page for DDE solvers, http://www.radford.edu/~thompson/webddes/.